Bapatla Engineering College

(Autonomous)

Department of Information Technology

18ITL33 - Operating Systems Lab Manual

(w.e.f. 2018-2019).

Bapatla Engineering College :: Bapatla

(Autonomous under Acharya Nagarjuna University)
BAPATLA - 522102 Guntur District, A.P., INDIA
www.becbapatla.ac.in.

Contents

1. Lab Course Description

1. Course code :18ITL33
2. Course Title : **Operating Systems Lab**
3. Core/ Elective : Core
4. Pre-requisites (if any) : Problem Solving with Programming(C Programming)
5. Semester / Year in which offered : **II year III Semester**
6. No. of weeks of Instruction : 15 weeks
7. No. of hours per week : 3
8. **Course objectives:**

Students will be able to

CO1: Have a thorough understanding of the fundamentals of Operating Systems.

CO2:Learn the mechanisms of OS to handle processes and threads and their communication

CO3 : Learn the mechanisms involved in memory management in contemporary OS

CO4 : Gain knowledge on Mutual exclusion algorithms, deadlock detection algorithms

CO5 : Know the components and management aspects of concurrency management

CO6 : Gain knowledge on file I/O operations and protection of various OS.

9. **Course outcomes:**

After the course the students are expected to be able to

CLO1: Understand different structures, services of the operating system and the use of scheduling and operations on process.

CLO2: Understand the use of scheduling, operations on process, the process scheduling algorithms and synchronization concepts.

CLO3: Understand the concepts of deadlock, memory and virtual memory management techniques.

CLO4: Understand the concepts of File System, Input/output systems and system protection of various operating systems.

10. **List of programs:** Separate sheet has been attached.
11. **Evaluation procedure**

INDEX	*EVALUATION*	*TOTAL MARKS*
A	*Marks allotted for day-to-day lab work .Max : 10 M per 1 program*	*20 M*
B	*Marks allotted for record Submission Max : 10 M per 1 program*	*15 M*
C	*Marks Awarded for Lab Examination*	*15 M*
D	*Continuous Internal Evaluation (CIE) (A+B+C)*	*50 M*
E	*Semester End Examination (SEE)*	*50 M*

2. List of Experiments

3. Lab Programs

1. Write a program to simulate FCFS Scheduling Algorithm to find turnaround time and waiting time.

```
//FCFS CPU SCHEDULING ALGORITHM
#include<stdio.h>
#include<conio.h>
 main()
 {
        int bt[20], wt[20], tat[20],
        i, n; float wtavg, tatavg;
        clrscr();
        printf("\nEnterthenumberofprocesses--");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nEnterBurstTimeforProcess%d--",i);
                scanf("%d",&bt[i]);
        }
        wt[0] = wtavg =
        0;
        tat[0]=tatavg=bt[0
        ];
         for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1]+bt[i-1];
                tat[i] = tat[i-1] +bt[i];
                wtavg=wtavg+wt[i];
                tatavg=tatavg+tat[i];
        }
        printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
        for(i=0;i<n;i++)
                printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
        printf("\nAverage Waiting Time -- %f", wtavg/n);
        printf("\nAverage Turnaround Time -- %f", tatavg/n);
        getch();
}
```

```
INPUT
Enter the number of processes--    3
Enter Burst Time for Process0--    24
Enter Burst Time for Process1--    3
Enter Burst Time for Process2--    3

OUTPUT
PROCESS        BURST TIME              WAITING TIME        TURNAROUND TIME
P0                  24                      0                   24
P1                  3                       24                  27
P2                  3                       27                  30


Average Waiting Time--      17.000000
Average Turnaround Time--   27.000000
```

2. Write a program to simulate SJF-non pre-emptive Scheduling Algorithm to find turnaround time and waiting time.

SJF CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<conio.h>
 main()
 {
        intp[20],bt[20],wt[20],tat[20],i,k,n,temp;
        float wtavg,tatavg;
        clrscr();
        printf("\nEnterthenumberofprocesses");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                p[i]=i;
                printf("EnterBurstTimeforProcess%d--",i);
                scanf("%d",&bt[i]);

        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(bt[i]>bt[k])
                        {
                                temp=bt[i];
                                bt[i]=bt[k];
                                bt[k]=temp;

                                   temp=p[i];
                                   p[i]=p[k];
                                   p[k]=temp;
                        }
        wt[0] = wtavg = 0;
        tat[0]=tatavg=bt[0];
        for(i=1;i<n;i++)
         {

                wt[i] = wt[i-1]+bt[i-1];
                tat[i] = tat[i-1] +bt[i];
                wtavg=wtavg+wt[i];
                tatavg=tatavg+tat[i];
        }
        printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND
        TIME\n");
        for(i=0;i<n;i++)
              printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i],tat[i]);
              printf("\nAverage Waiting Time -- %f", wtavg/n);
        printf("\nAverage Turnaround Time -- %f", tatavg/n);
        getch();
}
```

INPUT

Enter the number of processes-- 4
Enter Burst Time for Process0-- 6
Enter Burst Time for Process1-- 8
Enter Burst Time for Process2-- 7
Enter Burst Time for Process3-- 3

OUTPUT

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time-- 7.000000
Average Turnaround Time-- 13.000000

3. Write a program to simulate Priority-non pre-emptive Scheduling Algorithm to find turnaround time and waiting time.

```
//ROUNDROBINCPUSCHEDULINGALGORITHM
#include<stdio.h>
main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max; float awt=0,att=0,temp=0;
        clrscr();
        printf("Enterthenoofprocesses--");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                printf("\nEnterBurstTimeforprocess%d--",i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
        printf("\nEnterthesizeoftimeslice--");
        scanf("%d",&t);
        max=bu[0];
        for(i=1;i<n;i++)
                if(max<bu[i])
                        max=bu[i];
        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0)
                                if(bu[i]<=t)
                                {
                                        tat[i]=temp+bu[i];
                                        temp=temp+bu[i];
                                        bu[i]=0;
                                        }

                                else
                                  {
                                        bu[i]=bu[i]-t;
                                        temp=temp+t;
                                  }

        for(i=0;i<n;i++){
                wa[i]=tat[i]-ct[i];
                 att+=tat[i];
                awt+=wa[i];
        }
        printf("\nThe Average Turnaround time is -- %f",att/n);
        printf("\nThe Average Waiting time is -- %f ",awt/n);
        printf("\n\tPROCESS\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
        for(i=0;i<n;i++)
                printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
        getch();
```

```
}
```

INPUT

```
Enter the no of processes – 3
EnterBurstTimeforprocess1–  24
EnterBurstTimeforprocess2--  3
EnterBurstTimeforprocess3--  3

Enter the size of time slice – 3
```

OUTPUT

```
The Average Turnaround time is–15.666667
 The Average Waiting time is--       5.666667
```

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	24	6	30
2	3	4	7
3	3	7	10

4. Write a program to simulate Round Robin Scheduling Algorithm to find turnaround time and waiting time.

```
//PRIORITY CPU SCHEDULINGALGORITHM
#include<stdio.h>
main()
{
        Int p[20],bt[20],pri[20],wt[20],tat[20],i,k,n,temp;
        float wtavg,tatavg;
        clrscr();
        printf("Enterthenumberofprocesses---");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("EntertheBurstTime&PriorityofProcess%d---",i);
                scanf("%d %d",&bt[i],&pri[i]);
        }
        for(i=0;i<n;i++)
              for(k=i+1;k<n;k++)
                        if(pri[i] > pri[k])
                        {
                                temp=p[i]; p[i]=p[k]; p[k]=temp;
                                temp=bt[i]; bt[i]=bt[k]; bt[k]=temp;
                                temp=pri[i]; pri[i]=pri[k]; pri[k]=temp;
                        }
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];
                wtavg=wtavg+wt[i];
                tatavg=tatavg+tat[i];
        }

        printf("\nPROCESS\t\tPRIORITY\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIM
        E");
        for(i=0;i<n;i++)
               printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        getch();
}
```

INPUT

Enter the number of processes--5

Enter the Burst Time & Priority of Process0---10 3

Enter the Burst Time & Priority of Process1---1 1

Enter t he Burst Time & Priority of Process2---2 4

Enter the Burst Time & Priority of Process3---1 5

Enter the Burst Time & Priority of Process4---5 2

OUTPUT PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Average Waiting Time is --- 8.200000

Average Turnaround Time is---12.000000

5. Write a Program to simulate the concept of Dining-Philosophers problem.

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N
int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
sem_t mutex;
sem_t S[N];
void test(int phnum)
{
        if (state[phnum] == HUNGRY
                && state[LEFT] != EATING
                && state[RIGHT] != EATING) {
                // state that eating
                state[phnum] = EATING;
                sleep(2);
                printf("Philosopher %d takes fork %d and %d\n",
                                phnum + 1, LEFT + 1, phnum + 1);
                printf("Philosopher %d is Eating\n", phnum + 1);
                // sem_post(&S[phnum]) has no effect
                // during takefork
                // used to wake up hungry philosophers
                // during putfork
                sem_post(&S[phnum]);
        }
}
```

```
// take up chopsticks
void take_fork(int phnum)
{
        sem_wait(&mutex);
        // state that hungry
        state[phnum] = HUNGRY;
        printf("Philosopher %d is Hungry\n", phnum + 1);
        // eat if neighbours are not eating
        test(phnum);
        sem_post(&mutex);
        // if unable to eat wait to be signalled
        sem_wait(&S[phnum]);
        Sleep(1);
}
// put down chopsticks
void put_fork(int phnum)
{
        sem_wait(&mutex);
        // state that thinking
        state[phnum] = THINKING;
        printf("Philosopher %d putting fork %d and %d down\n",
        phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is thinking\n", phnum + 1);
        test(LEFT);
        test(RIGHT);
        sem_post(&mutex);
}
void* philospher(void* num)
{
        while (1) {

                int* i = num;
```

```
                sleep(1);
                take_fork(*i);
                sleep(0);
                put_fork(*i);
        }
}
int main()
{
        int i;
        pthread_t thread_id[N];
        // initialize the semaphores
        sem_init(&mutex, 0, 1);

        for (i = 0; i < N; i++)
                sem_init(&S[i], 0, 0);
        for (i = 0; i < N; i++) {
                // create philosopher processes
                pthread_create(&thread_id[i], NULL,philospher, &phil[i]);
                printf("Philosopher %d is thinking\n", i + 1);
        }
        for (i = 0; i < N; i++)
                pthread_join(thread_id[i], NULL);
}
```

OUTPUT:

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 1 is Hungry
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 4 is Hungry
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
```

6. Write a program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while(1)
{
printf("\nEnter your choice:");
scanf("%d",&n);
switch(n)
{
case 1: if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!");
break;
case 2: if((mutex==1)&&(full!=0))
consumer();
else
printf("Buffer is empty!!");
break;
case 3:
exit(0);
break;
}
}
```

```
return 0;
}
int wait(int s)
{
return (--s);
}

int signal(int s)
{
return(++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nProducer produces the item %d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\nConsumer consumes item %d",x);
x--;
mutex=signal(mutex);
}
```

OUTPUT

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3
```

7. Write a program to simulate Bankers Algorithm for deadlock avoidance.

```
#include <stdio.h>

#include <stdlib.h>

int main()
{
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
int p, r, i, j, process, count;
count = 0;

printf("Enter the no of processes : ");
scanf("%d", &p);

for(i = 0; i< p; i++)
completed[i] = 0;

printf("\n\nEnter the no of resources : ");
scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ", i + 1);
for(j = 0; j < r; j++)
scanf("%d", &Max[i][j]);
}

printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
printf("\nFor process %d : ",i + 1);
for(j = 0; j < r; j++)
scanf("%d", &alloc[i][j]);
}

printf("\n\nEnter the Available Resources : ");
for(i = 0; i < r; i++)
scanf("%d", &avail[i]);

for(i = 0; i < p; i++)

for(j = 0; j < r; j++)
need[i][j] = Max[i][j] - alloc[i][j];

do
{
printf("\n Max matrix:\tAllocation matrix:\n");

for(i = 0; i < p; i++)
{
```

```
for( j = 0; j < r; j++)
printf("%d ", Max[i][j]);
printf("\t\t");
for( j = 0; j < r; j++)
printf("%d ", alloc[i][j]);
printf("\n");
}

process = -1;

for(i = 0; i < p; i++)
{
if(completed[i] == 0)//if not completed
{
process = i ;
for(j = 0; j < r; j++)
{
if(avail[j] < need[i][j])
{
process = -1;
break;
}
}
}
if(process != -1)
break;
}

if(process != -1)
{
printf("\nProcess %d runs to completion!", process + 1);
safeSequence[count] = process + 1;
count++;
for(j = 0; j < r; j++)
{
avail[j] += alloc[process][j];
alloc[process][j] = 0;
Max[process][j] = 0;
completed[process] = 1;
}
}
}
while(count != p && process != -1);

if(count == p)
{
printf("\nThe system is in a safe state!!\n");
printf("Safe Sequence : <");
for( i = 0; i < p; i++)
printf("%d ", safeSequence[i]);
printf(">\n");
```

```
}
else
printf("\nThe system is in an unsafe state!!");

}
```

OUTPUT

```
nter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :
For process 1 : 7
5
3

For process 2 : 3
2
2

For process 3 : 7
0
2

For process 4 : 2
2
2

For process 5 : 4
3
3


Enter the allocation for each process :
For process 1 : 0
1
0

For process 2 : 2
0
0

For process 3 : 3
0
2

For process 4 : 2
1
1
```

```
For process 5 : 0
0
2


Enter the Available Resources : 3
3
2

Max matrix:    Allocation matrix:
7 5 3          0 1 0
3 2 2          2 0 0
7 0 2          3 0 2
2 2 2          2 1 1
4 3 3          0 0 2

Process 2 runs to completion!
Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
7 0 2          3 0 2
2 2 2          2 1 1
4 3 3          0 0 2

Process 3 runs to completion!
Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
2 2 2          2 1 1
4 3 3          0 0 2

Process 4 runs to completion!
Max matrix:    Allocation matrix:
7 5 3          0 1 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2

Process 1 runs to completion!
Max matrix:    Allocation matrix:
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
0 0 0          0 0 0
4 3 3          0 0 2

Process 5 runs to completion!
The system is in a safe state!!
Safe Sequence : < 2 3 4 1 5 >
```

8. Write a program to simulate Deadlock Detection algorithm.

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;

int main()
{
int alloc[10][10],request[10][10],avail[10],r[10],w[10];

printf("\nEnter the no of process: ");
scanf("%d",&np);
printf("\nEnter the no of resources: ");
scanf("%d",&nr);
for(i=0;i<nr;i++)
{
printf("\nTotal Amount of the Resource R%d: ",i+1);
scanf("%d",&r[i]);
}
printf("\nEnter the request matrix:");

for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&request[i][j]);

printf("\nEnter the allocation matrix:");
for(i=0;i<np;i++)
for(j=0;j<nr;j++)
scanf("%d",&alloc[i][j]);
/*Available Resource calculation*/
for(j=0;j<nr;j++)
{
avail[j]=r[j];
for(i=0;i<np;i++)
{
avail[j]-=alloc[i][j];

}
}

//marking processes with zero allocation

for(i=0;i<np;i++)
{
int count=0;
 for(j=0;j<nr;j++)
   {
     if(alloc[i][j]==0)
       count++;
     else
```

```
        break;
     }
 if(count==nr)
 mark[i]=1;
}
// initialize W with avail

for(j=0;j<nr;j++)
    w[j]=avail[j];

//mark processes with request less than or equal to W
for(i=0;i<np;i++)
{
int canbeprocessed=0;
 if(mark[i]!=1)
{
   for(j=0;j<nr;j++)
    {
      if(request[i][j]<=w[j])
        canbeprocessed=1;
      else
         {
         canbeprocessed=0;
         break;
          }
     }
if(canbeprocessed)
{
mark[i]=1;

for(j=0;j<nr;j++)
w[j]+=alloc[i][j];
}
}
}

//checking for unmarked processes
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;


if(deadlock)
printf("\n Deadlock detected");
else
printf("\n No Deadlock possible");
}
```

OUTPUT:

```
Enter the no of process: 4
Enter the no of resources: 5

Total Amount of the Resource R1: 2
Total Amount of the Resource R2: 1
Total Amount of the Resource R3: 1
Total Amount of the Resource R4: 2
Total Amount of the Resource R5: 1

Enter the request matrix:0 1 0 0 1
0 0 1 0 1
0 0 0 0 1
1 0 1 0 1

Enter the allocation matrix:1 0 1 1 0
1 1 0 0 0
0 0 0 1 0
0 0 0 0 0

 Deadlock detected
```

9. Write a program to simulate FIFO page replacement algorithms.

```
#include<stdio.h>

int main()
{
  int i,j,n,a[50],frame[10],no,k,avail,count=0;
  printf("\n ENTER THE NUMBER OF PAGES:\n");
  scanf("%d",&n);
 printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&no);
 for(i=0;i<no;i++)
frame[i]= -1;
   j=0;
printf("\tref string\t page frames\n");
  for(i=1;i<=n;i++)
        {
                printf("%d\t\t",a[i]);
                avail=0;
                for(k=0;k<no;k++)
                if(frame[k]==a[i])
                        avail=1;
                if (avail==0)
                {
                        frame[j]=a[i];
                        j=(j+1)%no;
                        count++;
                        for(k=0;k<no;k++)
                        printf("%d\t",frame[k]);
                }
                        printf("\n");
        }
        printf("Page Fault Is %d",count);
        return 0;
}
```

OUTPUT:

ENTER THE NUMBER OF PAGES: 20

ENTER THE PAGE NUMBER : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

ENTER THE NUMBER OF FRAMES :

ref string	page frames		
7	7	-1	-1
0	7	0	-1
1	7	0	1
2	2	0	1
0			
3	2	3	1
0	2	3	0
4	4	3	0
2	4	2	0
3	4	2	3
0	0	2	3
3			
2			
1	0	1	3
2	0	1	2
0			
1			
7	7	1	2
0	7	0	2
1	7	0	1

Page Fault Is 15

10. Write a program to simulate LRU page replacement algorithms.

```
#include<stdio.h>

main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:");
scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
        scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
        {
                c1=0;
                for(j=0;j<f;j++)
                {
                        if(p[i]!=q[j])
                        c1++;
                }
                if(c1==f)
                {
                        c++;
                        if(k<f)
                        {
                                q[k]=p[i];
                                k++;
                                for(j=0;j<k;j++)
                                printf("\t%d",q[j]);
                                printf("\n");
                        }
                        else
                        {
                                for(r=0;r<f;r++)
                                {
                                        c2[r]=0;
                                        for(j=i-1;j<n;j--)
                                        {
                                        if(q[r]!=p[j])
                                        c2[r]++;
                                        else
                                        break;
                                }
```

```
                        }
                        for(r=0;r<f;r++)
                         b[r]=c2[r];
                        for(r=0;r<f;r++)
                        {
                                for(j=r;j<f;j++)
                                {
                                        if(b[r]<b[j])
                                        {
                                                t=b[r];
                                                b[r]=b[j];
                                                b[j]=t;
                                        }
                                }
                        }
                        for(r=0;r<f;r++)
                        {
                                if(c2[r]==b[0])
                                q[r]=p[i];
                                printf("\t%d",q[r]);
                        }
                        printf("\n");
                }
        }
}
printf("\nThe no of page faults is %d",c);
}
```

OUTPUT:

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1

Enter no of frames:3

```
7
7   5
7   5   9
4   5   9
4   3   9
4   3   7
9   3   7
9   6   7
9   6   2
1   6   2
```

The no of page faults is 10

11. Write a program to simulate OPR page replacement algorithms.

```
include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);
   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);
   printf("Enter page reference string: ");
   for(i = 0; i < no_of_pages; ++i){
      scanf("%d", &pages[i]);
   }
   for(i = 0; i < no_of_frames; ++i){
      frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i){
      flag1 = flag2 = 0;
      for(j = 0; j < no_of_frames; ++j){
         if(frames[j] == pages[i]){
               flag1 = flag2 = 1;
               break;
            }
      }
      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
               faults++;
               frames[j] = pages[i];
               flag2 = 1;
```

```
            break;
          }
       }
    }
if(flag2 == 0)
    {
 flag3 =0;
    for(j = 0; j < no_of_frames; ++j){
     temp[j] = -1;
     for(k = i + 1; k < no_of_pages; ++k)
     {
     if(frames[j] == pages[k]){
                                  temp[j] = k;
                                  break;
                                   }
     }
     }

    for(j = 0; j < no_of_frames; ++j){
     if(temp[j] == -1){
     pos = j;
     flag3 = 1;
     break;
     }
     }
    if(flag3 ==0){
     max = temp[0];
     pos = 0;
     for(j = 1; j < no_of_frames; ++j){
     if(temp[j] > max){
     max = temp[j];
     pos = j;
     }
```

```
            }
            }

frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

OUTPUT

```
Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 2 3 4 2 1 3 7 5 4 3

2 -1 -1
2 3 -1
2 3 4
2 3 4
1 3 4
1 3 4
7 3 4
5 3 4
5 3 4
5 3 4
```

12. Write a program to simulate the following Contiguous Memory Allocation techniques:

    a. worst-fit
    b. best-fit
    c. first-fit

a)Worst-Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{
ff[i]=j;
break;
}
}
}
```

```
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

INPUT
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	1	5	4
2	4	3	7	3

b) Best-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
if(lowest>temp)
{
ff[i]=j;

lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
```

```
}
```

INPUT
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	2	2	1
2	4	1	5	1

c) First-fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{

for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
```

```
getch();
}
```

INPUT

```
Enter the number of blocks: 3
Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files:-
File 1: 1
File 2: 4
```

OUTPUT

File No	File Size	Block No	Block Size	Fragment
1	1	3	7	6
2	4	1	5	1

13. Implement Paging technique of memory management.

```
#include<stdio.h>
void main()
{
int memsize=15;
int pagesize,nofpage;
int p[100];
int frameno,offset;
int logadd,phyadd;
int i;
int choice=0;
printf("\nYour memsize is %d ",memsize);
printf("\nEnter page size:");
scanf("%d",&pagesize);
nofpage=memsize/pagesize;
for(i=0;i<nofpage;i++)
{
printf("\nEnter the frame of page%d:",i+1);
scanf("%d",&p[i]);
}
do
{
printf("\nEnter a logical address:");
scanf("%d",&logadd);
frameno=logadd/pagesize;
offset=logadd%pagesize;
phyadd=(p[frameno]*pagesize)+offset;
printf("\nPhysical address is:%d",phyadd);
printf("\nDo you want to continue(1/0)?:");
scanf("%d",&choice);
}while(choice==1);
}
```

OUTPUT:

```
Your memsize is 15
Enter page size:5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address:3

Physical address is:13
Do you want to continue(1/0)?:1
Enter a logical address:1

Physical address is:11
Do you want to continue(1/0)?:0
```